|  |
| --- |
| **课程论文成绩：** |
| 评语：  任课教师签字：  年 月 日 |

工程基础与创新设计

基于机器视觉的刹车片背板质检系统设计

2023级，电子信息，邵嘉懿，孙中海，周晓昀

摘要：基于图像处理的汽车刹车片的检测能够提高检测效率,本文使用连通域检测完成刹车片与背景分割，利用形态学处理和连通区域计算得到表面裂痕,能够实现对有裂纹缺陷的刹车片的检测.设计了一套基于OpenCV的检测背板关键尺寸数值，识别并检测背板的字符与缺陷的质检系统.并编写了基于wxWidgets的GUI界面.

关键词: 刹车片背板；缺陷检测；OpenCV；wxWidgets

# 引言

课题要求：1.自动检测背板关键尺寸数值；2.自动识别并检测背板的字符；3.自动识别并检测背板的铁锈与划痕；4.编写上位机图像处理的软件界面。

对于要求1、2和3，将基于一些图像处理的相关工具，来识别被检测量的几何特征，并以此来进行下一步的提取、分类等工作；对于要求4，基于开源的跨平台C++架构库实现GUI界面开发。

# 方案选择

方案一： 基于YOLO的深度学习图像处理工具，完成刹车片的质检工作。

方案二： 基于OpenCV的跨平台[计算机视觉](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E8%A7%86%E8%A7%89/2803351" \t "https://baike.baidu.com/item/opencv/_blank)和[机器学习](https://baike.baidu.com/item/%E6%9C%BA%E5%99%A8%E5%AD%A6%E4%B9%A0/217599)软件库，完成刹车片的质检工作。

综合比较以上两个方案，我们选取方案二作为本次课题的设计思路。原因在于，方案一的深度学习需要大量的数据集作为系统的训练材料，在刹车片缺陷图片较少的前提下，这个方案难以实现。其次，相对而言，方案二的程序运行花费的时间较少，更方便之后的调试与修改。

# 距离检测

## 图像预处理

对于计算机的屏幕设备而言，像素(Pixel)或者说PX是一个最基本的单位，就是一个点。其它所有的单位，都和像素成一个固定的比例换算关系。所有的长度单位基于屏幕进行显示的时候，都统一先换算成为像素的多少，然后进行显示。

从图片中截取尺子1cm的长度，通过opencv库中的相关函数，可以得到这1cm图片的长度像素值，然后通过 1/高度的像素值，可以计算得到通过手机拍摄得到的图片中一个像素值等于实际的多少长度，这边计算得到的数值是：每个像素对应 0.00421627cm。将图像灰度化处理如图2所示，从直方图分布中可以看出背景与刹车片主体之间的分布明显，如图3所示。使用threshold函数将灰度图转换为二值图，其中低阈值设置为130，高阈值设置为255。

通过查找轮廓操作，得到刹车片的外部轮廓并将外部轮廓用白色画在黑色背景图片上。

|  |  |
| --- | --- |
|  |  |
| 图1：比例尺计算用图 | 图2：长度计算灰度图 |
|  |  |
| 图3：灰度化直方图 | 图4：刹车片轮廓图 |

## 羊角距离检测

检测刹车片的尺寸需要寻找图像之间的特征，建立对应关系点，也就是图像中的特殊位置，是很常用的一类特征。点的局部特征也可以叫做“关键特征点”(keypoint feature)，或“兴趣点”(interest point)，或“角点”(conrner)。

关于角点的具体描述可以有几种：

1. 一阶导数(即灰度的梯度)的局部最大所对应的像素点；
2. 两条及两条以上边缘的交点；
3. 图像中梯度值和梯度方向的变化速率都很高的点；
4. 角点处的一阶导数最大，二阶导数为零，指示物体边缘变化不连续的方向。

Harris角点检测: 当一个窗口在图像上移动，在平滑区域如图(a)，窗口在各个方向上没有变化。在边缘上如图(b)，窗口在边缘的方向上没有变化。在角点处如图(c)，窗口在各个方向上具有变化，如图5所示。

Harris角点检测正是利用了这个直观的物理现象，通过窗口在各个方向上的变化程度，决定是否为角点。

|  |
| --- |
|  |
| 图5 Harris角点检测 |

Shi-Tomasi算法：Shi-Tomasi 算法是Harris 算法的改进。Harris 算法最原始的定义是将矩阵 M 的行列式值与 M 的迹相减，再将差值同预先给定的阈值进行比较。后来Shi 和Tomasi 提出改进的方法，若两个特征值中较小的一个大于最小阈值，则会得到强角点。

如图5(b)所示，对自相关矩阵M进行特征值分析，产生两个特征值（）和两个特征方向向量。因为较大的不确定度取决于较小的特征值，也就是（），所以通过寻找最小特征值的最大值来寻找好的特征点也就解释的通了。Shi和Tomasi 的方法比较充分，并且在很多情况下可以得到比使用Harris 算法更好的结果。

因此这里选用Shi-Tomasi算法，来检测刹车片关键点间的距离。此时直接进行角点检测会把很多不需要的角点检测到，所以从中选择需要测距的主要范围，然后通过Shi-Tomasi角点检测并筛选出我们需要点的坐标，计算两点间直线的像素距离，并通过之前计算得到的每个像素值代表的实际距离，可以得到刹车片关键尺寸的实际距离为13.23cm。最终输出图像如图6所示。相关代码如下：

1. DistanceResult calculate\_length(**const** cv::Mat &image) {
2. DistanceResult result;
3. // 灰度化
4. cv::Mat img\_gray;
5. cv::cvtColor(image, img\_gray, cv::COLOR\_BGR2GRAY);
6. // 二值化
7. cv::Mat img\_thresh1;
8. cv::threshold(img\_gray, img\_thresh1, 130, 255, cv::THRESH\_BINARY);
9. // 查找轮廓
10. std::vector<std::vector<cv::Point>> contours;
11. cv::findContours(img\_thresh1, contours, cv::RETR\_TREE, cv::CHAIN\_APPROX\_SIMPLE);
12. // 选择刹车片外部的轮廓
13. std::vector<cv::Point> cnt = contours[1];
14. // 创建灰度图那样大小的黑色底图
15. cv::Mat draw\_img = cv::Mat::zeros(img\_gray.size(), CV\_8UC1);
16. cv::drawContours(draw\_img, std::vector<std::vector<cv::Point>>{cnt}, 0, cv::Scalar(255), 2);
17. // 将匹配到的刹车片模板上下关键尺寸位置大致提取出来
18. cv::Mat draw\_img\_ROI = draw\_img(cv::Rect(277, 1100, 3753 - 277, 1600 - 1100));
19. // 创建原始图像的 ROI
20. cv::Mat image\_ROI = image(cv::Rect(277, 1100, 3753 - 277, 1600 - 1100)).clone(); // 克隆以避免修改原图像
21. // Shi-Tomasi角点检测
22. std::vector<cv::Point2f> corners;
23. cv::goodFeaturesToTrack(draw\_img\_ROI, corners, 10, 0.01, 10);
24. // 检查角点数量以避免越界
25. **if** (corners.size() < 10) {
26. std::cerr << "do not have 10" << std::endl;
27. result.distance = 0;
28. result.result\_image = image.clone(); // 返回原图像
29. **return** result;
30. }
31. // 获取上部和下部的关键点
32. cv::Point top\_point = corners[9];
33. cv::Point down\_point = corners[8];
34. // 把点画出来
35. cv::circle(image\_ROI, top\_point, 10, cv::Scalar(0, 0, 255), -1);
36. cv::circle(image\_ROI, down\_point, 10, cv::Scalar(0, 0, 255), -1);
37. // 计算距离
38. result.distance = two\_point\_distance(top\_point, down\_point);
39. // 绘制线条
40. cv::line(image\_ROI, top\_point, down\_point, cv::Scalar(0, 0, 255), 2);
41. // 将距离显示在图像上
42. std::string distance\_text = "Distance: " + std::to\_string(result.distance) + " cm";
43. cv::putText(image\_ROI, distance\_text, cv::Point(50, 50), cv::FONT\_HERSHEY\_SIMPLEX, 2, cv::Scalar(0, 0, 255), 2);
44. // 将图像和距离保存到结果结构体中
45. result.result\_image = image.clone(); // 使用原始图像并克隆
46. // 将绘制的 ROI 图像粘贴回原图像
47. image\_ROI.copyTo(result.result\_image(cv::Rect(277, 1100, 3753 - 277, 1600 - 1100)));
48. // 返回结果结构体
49. **return** result;
50. }

|  |
| --- |
|  |
| 图6：羊角长度计算图 圆孔距离检测 圆孔距离的检测也是类似，唯一不同的是，在灰度图中圆的特征就已经十分明显了，因此并不需要二值化处理。转化成灰度图后再使用高斯模糊去除噪声，再使用HoughCircles检测出图中的圆，并在图中画出检测到的圆并计算距离。实现效果如图7所示。相关代码如下：   1. **void** CircleDetector::detectCircles(cv::Mat image) 2. { 3. cv::Mat gray; 4. cv::cvtColor(image, gray, cv::COLOR\_BGR2GRAY); 5. cv::GaussianBlur(gray, gray, cv::Size(9, 9), 2, 2); 7. cv::HoughCircles(gray, circles\_, cv::HOUGH\_GRADIENT, 1, 8. gray.rows /5,  // change this value to detect circles with different distances to each other 9. 100, 30, 10, 200 // change the last two parameters 10. ); 11. // 过滤检测到的圆 12. std::vector<cv::Vec3f> filteredCircles; 13. **for** (**const** auto& circle : circles\_) { 14. cv::Point center(cvRound(circle[0]), cvRound(circle[1])); 15. **int** radius = cvRound(circle[2]); 16. // 过滤条件：半径适中 17. **if** (radius > 100 && radius < 145) { 18. filteredCircles.push\_back(circle); 19. } 20. } 21. circles\_ = filteredCircles; 22. // 保存检测到的圆心位置 23. **if** (circles\_.size() >= 2) { 24. circleCenter1\_ = cv::Point(circles\_[0][0], circles\_[0][1]); 25. circleCenter2\_ = cv::Point(circles\_[1][0], circles\_[1][1]); 26. } 27. } 28. **void** CircleDetector::drawDetectedCircles(cv::Mat& image) { 29. **for** (**size\_t** i = 0; i < circles\_.size(); i++) { 30. cv::Vec3i c = circles\_[i]; 31. cv::Point center = cv::Point(c[0], c[1]); 32. // circle center 33. cv::circle(image, center, 1, cv::Scalar(0, 100, 100), 3, cv::LINE\_AA); 34. // circle outline 35. **int** radius = c[2]; 36. cv::circle(image, center, radius, cv::Scalar(0, 0, 255), 3, cv::LINE\_AA); 37. } 38. // 绘制连接两个圆心的线 39. **if** (circleCenter1\_.x && circleCenter2\_.x) { 40. cv::line(image, circleCenter1\_, circleCenter2\_, cv::Scalar(0, 0, 255), 2); 41. // 计算两个圆心之间的距离 42. **double** distance = calculateDistance(circleCenter1\_, circleCenter2\_); 43. // 创建文本框显示距离 44. std::string distanceText = "Distance: " + std::to\_string(distance/calculatePixelsPerCm()) + " cm"; 45. cv::putText(image, distanceText, cv::Point(image.cols / 2, image.rows / 2), cv::FONT\_HERSHEY\_SIMPLEX, 3, cv::Scalar(0, 0, 255), 2); 46. } 47. }     图7：圆心距离计算图 |

# 字符识别

计算机文字识别，俗称光学字符识别，英文全称是Optical Charater Recognition（简称OCR）,它是利用光学技术和计算机技术把印刷在或者写在图纸上的文字以文本形式提取出来，并转换成一种计算机能够接受、人又可以理解的格式。OCR技术是实现文字快速录入的一项关键技术。在信息社会时代，每天会产生大量的票据、表单、证件数据，这些数据要电子化，需要利用OCR技术进行提取录入。在深度学习没有全面推广之前，大部分OCR识别都是基于传统的方法进行检测识别。在背景单一、数据场景简单的情况下，传统OCR一般都能达到好的效果，但在一些场景复杂、干扰多的情况下，识别效果不好，这个时候深度学习OCR就能体现出巨大的优势。

传统OCR基于图像处理（二值化、连通域分析、投影分析等）和统计机器学习（Adaboot、SVM），提取图片上的文本内容；用到的计算机视觉算法库主要有OpenCV、Halcon、VisionPro等，出于开源、多语言支持和方便易用的考虑，本次课程设计通常采用OpenCV进行开发。

传统OCR识别流程：按处理方式划分为三个阶段：预处理阶段、识别阶段和后处理阶段。首先文本定位，接着进行倾斜文本矫正，分割出单字后，使用人工特征HOG或者CNN特征，结合分类模型对单字进行识别，最后基于统计语言模型（如隐马尔科夫链，HMM）或者规则进行语义纠错。其中关键在于预处理阶段，预处理阶段的质量直接决定了最终的识别效果。

|  |
| --- |
|  |
| 图8 OCR文字识别 |

通过过滤算法，首先将手机拍摄的原始图片进行灰度化二值化操作以及形态学中的应用侵蚀和膨胀操作等，过滤出符合要求的字符轮廓，然后通过填充轮廓获取完整字符。此时图片为黑底白字样式，通过取非操作将黑底白字转变为白底黑字。由于刹车片工件水平放置导致工件上的字符为竖直方向无法通过OCR直接识别，因此上述操作后将图片顺时针顺转90°然后进行OCR识别。各个阶段的结果如图9所示。

|  |  |
| --- | --- |
| text |  |
| a、图像位置截取 | b、图像轮廓 |
|  |  |
| c、图像形态学操作 | d、图像反相 |
| 图9 OCR处理过程 | |

# 刹车片缺陷检测

## 铁锈检测

由于在灰度图中铁锈与物体表面的灰度值差别太小，这样不利于我们对铁锈的识别，所以先将图片转化到HSV空间，转换后的图片如图10所示：

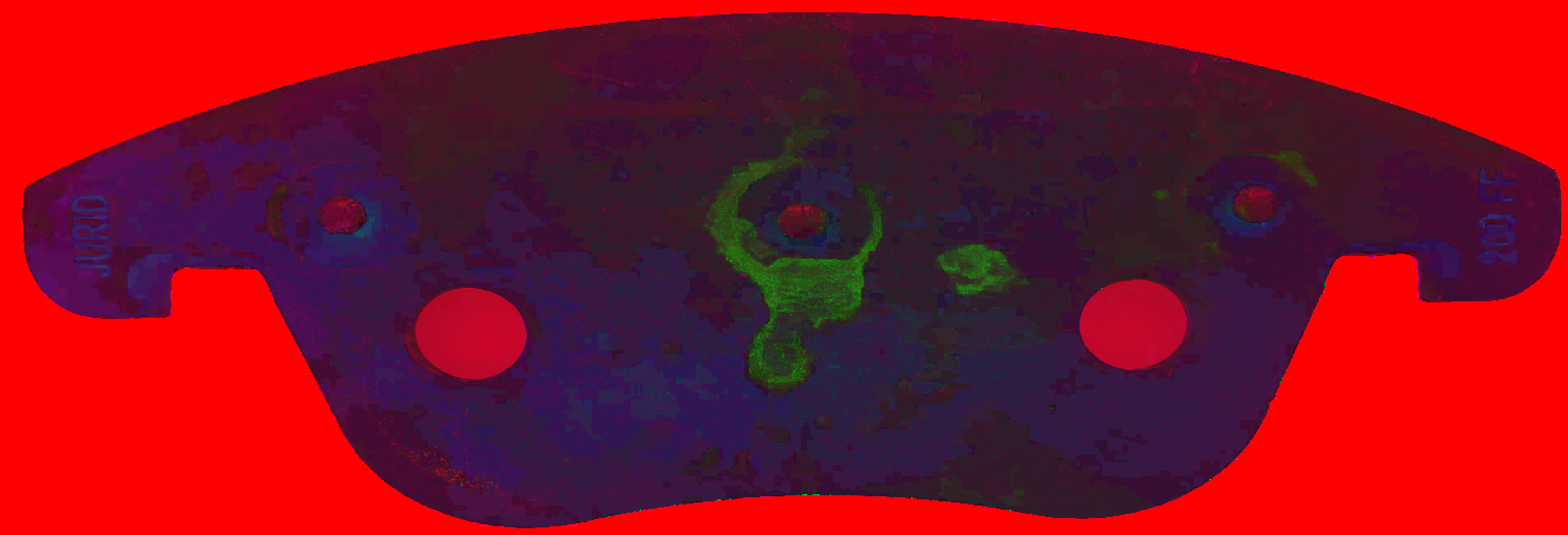


图10：刹车片HSV图

通过反复测试发现铁锈的HSV范围在(0, 46, 46)至(28, 255, 255)，于是将范围内的区域提取出来。随后进行形态学操作，去除噪声，并使铁锈区域连通。

最后通过findContour查找轮廓，由于细小铁锈点遍布表面，为了使输出的轮廓不受细小点的影响，通过面积筛选出合适的区域并在图上绘制出区域和面积文本，效果如图11所示。相关代码如下：

1. **void** detectRust(**const** cv::Mat& src, **double**& totalArea)
2. {
3. // 转换颜色空间到HSV。
4. cv::Mat hsv;
5. cv::cvtColor(src, hsv, cv::COLOR\_BGR2HSV);
7. // 设定颜色阈值，这里铁锈的颜色在H:0-28, S:46, V:46范围内
8. cv::Mat mask;
9. cv::inRange(hsv, cv::Scalar(0, 46, 46), cv::Scalar(28, 255, 255), mask);
11. // 形态学操作，去除噪声，并使铁锈区域连通
12. cv::Mat kernel = cv::getStructuringElement(cv::MORPH\_RECT, cv::Size(3, 3));
13. cv::morphologyEx(mask, mask, cv::MORPH\_OPEN, kernel);
14. cv::morphologyEx(mask, mask, cv::MORPH\_DILATE, kernel);
16. // 寻找轮廓
17. std::vector<std::vector<cv::Point>> contours;
18. cv::findContours(mask, contours, cv::RETR\_EXTERNAL, cv::CHAIN\_APPROX\_SIMPLE);
20. // 对轮廓进行面积过滤，剔除小面积的铁锈区域
21. std::vector<std::vector<cv::Point>> filteredContours;
22. **double** pixelToCmRatio = 0.00624; // 假设每个像素代表0.1厘米
23. totalArea = 0; // 用于存储总面积
24. **for** (**const** auto& contour : contours)
25. {
26. **double** area = cv::contourArea(contour);
27. **double** actualArea = area \* pixelToCmRatio \* pixelToCmRatio; // 将像素面积转换为实际面积
28. **if** (actualArea >= 0.09 && actualArea <= 5) // 根据需求调整面积阈值
29. {
30. filteredContours.push\_back(contour);
32. // 累加实际面积
33. totalArea += actualArea;
35. // 在图像上显示实际面积信息
36. std::string areaText = std::to\_string(actualArea) + " cm^2";
37. cv::putText(src, areaText, contour[0], cv::FONT\_HERSHEY\_SIMPLEX, 1, cv::Scalar(0, 255, 255), 2);
38. }
39. }
41. // 输出总面积
42. std::cout << "Total Area: " << totalArea << " cm^2" << std::endl;
44. // 在原图上绘制经过面积过滤的轮廓
45. cv::drawContours(src, filteredContours, -1, cv::Scalar(0, 255, 255), 2);
47. }



图11：铁锈检测图

## 划痕检测

首先还是读取为灰度图，由于在灰度图中划痕与工件表面的灰度值十分接近，这并不利于我们继续处理。需要对图像进行预处理使得两者的差距明显，将灰度范围为30-80的部分提取出来。然后对这一部分进行均衡化处理。使用阈值分割将图像二值化，对二值化图像进行膨胀操作减少噪声的干扰，再使用Canny边缘检测后执行概率霍夫直线检测检测出划痕，效果如图12所示。相关代码如下：

1. **void** processImage(**const** Mat& image, Mat& color\_image) {
2. // 将图像转换为灰度图像。
3. Mat gray;
4. cvtColor(image, gray, COLOR\_BGR2GRAY);
6. // 将灰度范围为30-80的部分提取出来
7. **int** min\_gray = 30;
8. **int** max\_gray = 80;
9. Mat masked\_image = Mat::zeros(gray.size(), CV\_8UC1);
10. **for** (**int** i = 0; i < gray.rows; ++i) {
11. **for** (**int** j = 0; j < gray.cols; ++j) {
12. **if** (gray.at<uchar>(i, j) >= min\_gray && gray.at<uchar>(i, j) <= max\_gray) {
13. masked\_image.at<uchar>(i, j) = gray.at<uchar>(i, j);
14. }
15. }
16. }
18. // 对提取出来的部分进行直方图均衡化
19. Mat equalized\_masked\_image;
20. equalizeHist(masked\_image, equalized\_masked\_image);
22. // 使用阈值分割将图像二值化
23. **int** threshold\_value = 70; // Renamed to avoid conflict
24. Mat output\_image;
25. threshold(gray, output\_image, threshold\_value, 255, THRESH\_BINARY);
27. // 对二值化图像进行膨胀操作
28. Mat kernel = getStructuringElement(MORPH\_RECT, Size(2, 2));
29. dilate(output\_image, output\_image, kernel, Point(-1, -1), 2);
31. // 使用Canny边缘检测
32. Mat edges;
33. Canny(output\_image, edges, 1, 300, 3);
35. // 执行概率霍夫直线检测
36. std::vector<Vec4i> lines;
37. HoughLinesP(edges, lines, 1, CV\_PI / 180, 1, 90, 8);
39. // 在彩色图像上绘制直线
40. **if** (!lines.empty()) {
41. **for** (**size\_t** i = 0; i < lines.size(); i++) {
42. Vec4i l = lines[i];
43. line(color\_image, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 2);
44. }
45. }
46. }



图12：缺陷检测结果示例

# 上位机软件设计

wxWidgets是一个跨平台的软件开发包。它最初是被设计成跨平台的GUI软件开发包，但后来随着越来越多的人参与进来，为wxWidgets加入了许多非GUI的功能，如多线程(MultiThread)、网络(Network)等。并且从最初的只支持C++语言，逐渐发展成为支持数种语言（如Python、Perl、C#、Basic等）。因此，现在的wxWidgets已经不再是单纯的跨平台的GUI软件开发包，而是一个可以支持多种操作系统平台的能够在多种语言中使用的通用跨平台软件开发包。

本次课程设计的上位机软件共包含4个功能。① 划痕检测：点击“划痕检测”按钮后检测的刹车片的划痕，并且在图像显示界面显示加载的图片。②距离检测：点击“距离检测”按钮后，在图像显示界面显示检测到的圆孔和两边羊角的图像，并在上位机主界面的文本框，即“圆孔距离:”和“羊角距离：”后面分别显示圆孔距离和羊角距离。③铁锈检测：点击“铁锈检测”按钮后在图像显示界面显示检测后的结果，并在“铁锈面积:”文本框后面显示铁锈总面积。④字符识别：点击“字符识别”按钮之后将检测结果显示“字符识别:”文本框后面。

|  |  |
| --- | --- |
|  |  |
| 1. 上位机主界面 |  |
| b、图像显示界面  图13：上位机界面 | |

# 结束语

本文设计了一套基于OpenCV的刹车片缺陷检测系统，其中主要包括了关键距离检测、OCR识别字符、铁锈与划痕检测功能，并在实验测试中顺利完成。此外，还设计了基于wxwidgets的上位机界面，提升了人机交互的体验，使得系统各部分模块有机整合，实现了操作运行更加方便快捷的需求。但是本次设计受限于时间与能力，还有许多可以改进的地方，比如：1、在羊角检测中，我们使用的是角点检测即检测羊角两个直角顶点，在实际中可能顶点会有所磕碰，导致测量不准，后续可以通过内侧直线进行测量距离。2、在本次设计中需要多次运用阈值分割实现图片二值化，怎么样快速确定合适的阈值还需要继续研究。

# 致谢

在这两周的时间里，我们小组的各成员均受益匪浅。负责距离检测和划痕的邵嘉懿同学、负责铁锈检测和OCR字符识别的孙中海同学自主学习了图像处理的相关知识，查阅了大量的相关文献，编写了相关程序，并最后取得了不错的检测效果。周晓昀同学则是编写了基于WxWidgets的上位机C++程序，使得系统各模块在调用起来更方便的同时，使得检测系统更易上手、更方便推广使用。

除此之外，王栋老师也对我们的工作提供了相当有效的帮助，尤其是老师搭建的实物检测平台，以及各种各样的实物刹车片，都可以说是解决了当时我们工作的燃眉之急，故在此特谢。

# 参考文献:

1. 韩志玮,高美凤.刹车片表面缺陷的图像检测方法[J].应用光学,2020,41(03):538-547.
2. 费成,黄影平,胡兴,慈文彦.基于机器视觉的汽车刹车片自动检测系统[J].光学仪器,2018,40(03):76-83.
3. 胡琳丽. 基于机器视觉的刹车片尺寸综合检测系统研究[D].中国计量大学,2018.
4. 米春风,卢琨,汪文艳,王兵.基于机器视觉的带钢表面缺陷检测研究进展[J].安徽工业大学学报(自然科学版),2022,39(02):180-188.
5. 高文彬,庄申乐,王秀剑,周敏,宋冉冉,张成雷.基于机器视觉技术的壳体表面缺陷检测研究[J].机电元件,2022,42(01):42-46.
6. 曾强. 基于机器视觉的刹车片缺陷检测方法研究[D].中国计量大学,2017.
7. Xu Rongge,Hao Ruiyang,Huang Biqing. Efficient surface defect detection using self-supervised learning strategy and segmentation network[J]. Advanced Engineering Informatics,2022,52.
8. Wang Xiaofeng,Zhang Yanan,Liu Jun,et al. Online detection of weld surface defects based on improved incremental learning approach[J]. Expert Systems With Applications,2022,195.
9. Liu Yang,Yuan Yachao,Liu Jing. Deep learning model for imbalanced multi-label surface defect classification[J]. Measurement Science and Technology,2022,33(3).
10. Chen Xuechun,Lv Jun,Fang Yulun,et al. Online Detection of Surface Defects Based on Improved YOLOV3[J]. Sensors,2022,22(3).
11. Jiawei Lian,Junhong He,Yun Niu, et al. Fast and accurate detection of surface defect based on improved YOLOv4[J]. Assembly Automation,2022,42(1).